พอยน์เตอร์ในภาษาปาสคาล

บุญเลิศ เอี่ยมทัศนา

ของดีจุดเด่นในภาษาปาสคาลเรื่องหนึ่งคือพอยน์เตอร์ แต่ไม่ค่อยจะกล่าวถึงกัน ทั้งที่ความสำคัญของพอยน์เตอร์นั้นอาจกล่าวได้ว่าภาษาใดที่ไม่สามารถกำหนดตัวแปรแบบพอยน์เตอร์ได้ ภาษานั้นจะเป็นภาษาชั้นประถมเท่านั้น

เหตุที่ไม่ค่อยกล่าวถึงกันก็เพราะว่าในภาษาปาสคาลนั้น เราใช้ตัวแปรแบบพอยน์เตอร์กับข้อมูลที่เป็นโครงสร้างสลับซับซ้อน หรือเมื่อดำเนินการกับข้อมูลจำนวนมาก และไม่มีความจำเป็นใดที่จะต้องนำมาใช้กับข้อมูลธรรมดาอย่างเช่นในภาษาซี จึงเกินขอบเขตของเรื่องที่จะลงวารสาร แต่เมื่อมีผู้ถามไปผมจึงขอนำมากล่าวถึง

ตัวแปรแบบพอยน์เตอร์ (Pointer)

ตัวแปรแบบพอยน์เตอร์แตกต่างกับตัวแปรแบบอื่น ๆ กล่าวคือ ตัวแปรแบบอื่นนั้นเรากำหนดขึ้นเพื่อดำเนินกรรมวิธีกับมันโดยตรง แต่ตัวแปรแบบพอยน์เตอร์นั้นเรากำหนดขึ้นเพื่อใช้ชี้ไปยังตัวแปรหรือข้อมูลอื่น ๆ อีกทอดหนึ่ง

ผมขอขยายความสักเล็กน้อยคือ เมื่อเรากำหนดตัวแปรจะเป็นการกำหนดเนื้อที่ในหน่วยความจำสำหรับตัวแปรนั้น (โดยที่เนื้อที่ในหน่วยความจำที่กำหนดให้จะขึ้นกับแบบของข้อมูล) ซึ่งสำหรับตัวแปรแบบอื่น ๆ ถือว่าเป็นการกำหนดโดยตรงคือ เมื่อมีการดำเนินการใด ๆ กับตัวแปร จะเป็นการดำเนินการกับหน่วยความจำนี้โดยตรง

แต่สำหรับพอยน์เตอร์เมื่อกำหนดตัวแปรแล้วเรายังใช้งานมันไม่ได้ทันที จะต้องกำหนดต่อไปว่าจะให้มันชี้ไปยังข้อมูลคือหน่วยความจำใด โดยนำค่าแอดเดรสของหน่วยความจำนั้นมาให้แก่มัน ซึ่งหน่วยความจำนี้อาจมีการตั้งชื่อคือกำหนดตัวแปรแล้วหรือไม่ก็ได้ ต่อไปเมื่อเราดำเนินการใดกับพอยน์เตอร์ จะเป็นการดำเนินการกับหน่วยความจำนั้นผ่านพอยน์เตอร์อีกทอดหนึ่ง

การกำหนดตัวแปรพอยน์เตอร์

การกำหนดตัวแปรพอยน์เตอร์เพื่อดำเนินการกับข้อมูลใดก็ตาม เราต้องกำหนดตัวแปรพอยน์เตอร์ให้เป็นพอยน์เตอร์ของข้อมูลแบบนั้นด้วย เช่น

```
VAR arrow : ^Integer;
```

จะเป็นการกำหนดตัวแปร arrow เป็นตัวแปรแบบพอยน์เตอร์ สำหรับชี้ข้อมูลแบบเลขจำนวนเต็มเท่านั้นจะนำไปใช้ชี้ข้อมูลแบบอื่นไม่ได้ อนึ่ง ในการใช้งานจริงส่วนมากแล้ว ต้องกำหนดแบบของข้อมูลพอยน์เตอร์ขึ้นก่อน โดยเฉพาะเมื่อจะให้ชี้ไปยังข้อมูลที่เป็นโครงสร้าง ซึ่งก็แตกต่างกับข้อมูลแบบอื่นอีกคือ เราสามารถกำหนดแบบของพอยน์เตอร์ก่อนหรือหลังแบบของข้อมูลก็ได้ ดังแสดงในรูปที่ 1 กล่าวคือ

- ได้กำหนดแบบของข้อมูลแบบเรคอร์ดขึ้นก่อนให้ชื่อว่า stuRecT เสร็จแล้วได้กำหนดข้อมูลแบบพอยน์เตอร์เป็น studPntrT, จะเห็นว่าได้กำหนดแบบของพอยน์เตอร์ไว้หลังแบบของข้อมูล จากนี้จึงกำหนดตัวแปรพอยน์เตอร์ spntr เป็นพอยน์เตอร์สำหรับชี้ข้อมูลแบบ stuRecT เท่านั้น จะนำไปชี้ข้อมูลแบบอื่นไม่ได้
- ในกรณีของ nptrT กับ nameT ได้กำหนดแบบของข้อมูลพอยน์เตอร์ nptrT ก่อนที่จะกำหนดแบบของข้อมูลที่จะให้มันชี้คือ nameT จะเห็นว่าได้อ้าง nameT ก่อนที่จะกำหนดแบบของมัน ซึ่งปกติทำไม่ได้ จึงถือว่าเป็นกรณีพิเศษสำหรับพอยน์เตอร์

การใช้งานพอยน์เตอร์

การใช้งานพอยน์เตอร์แบ่งออกเป็น 2 แบบใหญ่ตามชนิดของข้อมูลคือ กับข้อมูลชนิดสแตติกส์ (statics) และกับข้อมูลชนิดไดนามิกส์ (dynamics)

ข้อมูลชนิดสแตติกส์คือข้อมูลที่คงที่แน่นอน เช่น เมื่อเรากำหนดตัวแปรแบบสตริง จะเป็นการกำหนดหน่วยความจำหน่วยหนึ่ง ขนาดตามจำนวนไบต์ของสตริง และให้ชื่อหน่วยความจำนี้ตามชื่อตัวแปรที่เราตั้ง ซึ่งหน่วยความจำนี้จะคงอยู่ตลอดการรันโปรแกรมนี้ (มีข้อยกเว้นสำหรับตัวแปรที่เป็นโลคอล แต่ถือว่าเป็นสแตติกส์เช่นกัน)

แต่สำหรับข้อมูลชนิดไดนามิกส์ เราไม่ได้กำหนดตัวแปรให้กับมัน หากกำหนดหน่วยความจำขึ้นหน่วยหนึ่ง แล้วให้พอยน์เตอร์ตัวใดตัวหนึ่งชี้ ซึ่งเราอาจจะเปลี่ยนให้พอยน์เตอร์ตัวอื่นมาชี้แทนเมื่อไรก็ได้ ไม่ใช่เป็นชื่อประจำเช่นตัวแปร สำหรับขนาดของหน่วยความจำก็มีข้อแตกต่างกันคือ เราสามารถจะกำหนดเป็นจำนวนไบต์ตามที่ต้องการก็ได้ หรือจะกำหนดตามแบบของข้อมูลแบบใดแบบหนึ่งก็ได้

อาจจะเห็นว่าขัดกับที่กล่าวในหัวข้อการกำหนดตัวแปรพอยน์เตอร์ เพราะตอนนั้นพูดถึงตัวแปรพอยน์เตอร์ แต่ตอนนี้พูดถึงข้อมูลแบบไดนามิกส์ หากยังคงสงสัยโปรดเก็บความสงสัยไว้ก่อนครับ และนอกจากนี้ยังมีพอยน์เตอร์ที่ไม่ใช่ตามแบบข้อมูลใดอีกด้วยครับ

ที่เรียกว่าเป็นข้อมูลแบบไดนามิกส์ เพราะว่าข้อมูลชนิดนี้เราจะสร้างเมื่อไรและยกเลิก (ไม่มีตัวตนอยู่ในหน่วยความจำ) เมื่อไรก็ได้ และจะสร้างซ้ำอีกเมื่อไรก็ได้ ซึ่งเมื่อสร้างซ้ำใหม่อาจจะอยู่ที่หน่วยความจำแอดเดรสเดิมหรือไม่ก็ได้ ขึ้นอยู่กับลักษณะของการใช้งาน

ตามที่กล่าวว่าแบ่งการใช้งานพอยน์เตอร์เป็น 2 แบบใหญ่ตามชนิดของข้อมูลนั้น แต่ข้อมูลแบบไดนามิกส์ยังอาจแบ่งออกได้เป็นอีก 2 แบบใหญ่ เพื่อสะดวกในการแบ่งหัวข้อจะขอแบ่งการใช้งานพอยน์เตอร์เป็น 3 แบบคือ

- กับตัวแปรสแตติกส์
- กับข้อมูลไดนามิกส์อิสระ
- กับข้อมูลไดนามิกส์โครงสร้าง

ซึ่งในฉบับนี้คงกล่าวได้เฉพาะ 2 หัวข้อแรก จึงขอขยายความเรื่องข้อมูลแบบไดนามิกส์ก่อน ซึ่งในหนังสือต่างประเทศจะกล่าวไว้ในหัวข้อ Dynamics data structure และก็ไม่ได้แบ่งย่อยอย่างที่ผมแบ่ง

ความหมายของผมคือ แบบอิสระนั้นข้อมูลแต่ละตัวไม่ได้มีความสัมพันธ์กัน หากจะคิดว่าสัมพันธ์กันก็สัมพันธ์กันในแบบการตากผ้าบนราว คือคนอาจจะเดินตามราวไปเก็บผ้าได้ แต่ถ้าผ้าผืนใดตกไม่ได้หมายความว่าจะทำให้ผ้าผืนอื่นตกไปด้วย ข้อมูลชนิดนี้ใช้มากในโปรแกรมประเภทเวิร์ดโปรเซสเซอร์ โดยที่ข้อมูลแต่ละบรรทัดที่เราป้อนจะเป็นข้อมูลตัวหนึ่ง ๆ

ส่วนแบบที่เป็นโครงสร้างนั้นข้อมูลแต่ละตัวจะมีความสัมพันธ์กัน ซึ่งขึ้นอยู่กับรูปแบบของโครงสร้าง แบบง่าย ๆ เทียบได้กับการผูกเชือกหลาย ๆ เส้นต่อกันแขวนถังไว้ หากเส้นใดเส้นหนึ่งขาด เส้นอื่น ๆ ทางด้านถังก็จะพลอยตกลงมาด้วย ข้อมูลแบบนี้และที่เป็นโครงสร้างซับซ้อนกว่านี้ เช่น เป็นแบบต้นไม้ (tree) ใช้กันมากในโปรแกรมประเภท ดาต้าเบสและสเปรดชีต

```
{ POINTER1 }

TYPE     StuRecT   = RECORD
           name    :  String[30];
           addr    :  String[50];
           class   :  String[4];
                      END;
         StudPntrT = ^StuRecT;
VAR      spntr     : StudPntrT;

TYPE     nptrT     =  ^nameT;
         nameT     =  String[35];
```

รูปที่ 1 การกำหนดตัวแปรพอยน์เตอร์

พอยน์เตอร์กับตัวแปร

ขอเรียนให้ทราบก่อนว่าการใช้งานพอยน์เตอร์กับตัวแปรในแบบนี้ ใช้กันมากในภาษาซีและจำเป็นต้องใช้ด้วย แต่ไม่จำเป็นต้องใช้ในภาษาปาสคาล จึงไม่นิยมใช้กันครับ แต่เพื่อให้เห็นข้อแตกต่างระหว่างตัวแปรพอยน์เตอร์กับตัวแปรแบบอื่น ๆ ที่เรียกว่าเป็นสแตติกส์ จึงขอนำมากล่าวถึงอย่างสั้น ๆ กล่าวคือ

เมื่อเรากำหนดตัวแปร เช่น ตามโปรแกรมตัวอย่าง PointerTwo (รูปที่ 2) คือ aname และ bname จะเป็นการกำหนดเนื้อที่ในหน่วยความจำขึ้น 2 หน่วยอย่างแน่นอน ดังที่ได้นำค่าแอดเดรสมาให้แก่พอยน์เตอร์ ptr1 และ ptr2 (เครื่องหมาย @ ทำหน้าที่เช่นเดียวกับฟังก์ชัน Addr คือให้ค่าแอดเดรสของตัวแปร แต่ @ ไม่มีในเทอร์โบปาสคาลรุ่นก่อนรุ่น 4.0)

ในโปรแกรมได้ให้ค่าคงตัวสตริง Alpha และ Bravo แก่ตัวแปร aname และ bname ตามลำดับ แต่ในคำสั่ง WriteLn ได้ใช้พอยน์เตอร์แทน ผลการรันถึงคำสั่งนี้จะปรากฏภาพบนจอภาพเป็น

Alpha Bravo

ทั้งนี้เพราะ ptr 1^ จะหมายถึงแอดเดรสของข้อมูล ซึ่งเมื่อผ่านค่าไปให้แก่โพรซีเดอร์ WriteLn จึงเป็นการนำข้อมูลนี้คือ Alpha ไปออกจอภาพ

ตัวอย่างต่อไปจะเปลี่ยน ptr1 ไปชี้ Bravo และให้ ptr2 ชี้ Alpha โดยนำตัวแปร tem มาเป็นตัวกลางเมื่อใช้คำสั่ง WriteLn ตามลำดับเดิม ผลการรันจะเป็น

Bravo Alpha

ความจริงในการเปลี่ยนตัวพอยน์เตอร์จะทำโดยตรงก็ได้คือ ptr1 : = Addr (bname) และ ptr2 = Addr (aname) ที่นำ tem มาเป็นตัวกลางเพื่อแสดงถึงพอยน์เตอร์อีกแบบหนึ่งคือพอยน์เตอร์ที่ไม่มีแบบข้อมูลกำกับขอให้ดูการกำหนดตัวแปรพอยน์เตอร์ tem ได้กำหนดเป็นแบบ Pointer เฉย ๆ ซึ่งใช้ในการเก็บแอดเดรสของข้อมูลได้ แต่จะใช้ชี้ข้อมูลคือ จะใช้ในแบบ tem^ ไม่ได้

อีกตัวอย่างหนึ่งแสดงการผ่านค่าแอดเดรสด้วยพอยน์เตอร์ไปให้แก่โพรซีเดอร์หรือฟังก์ชัน คือคำสั่ง Print (ptr1^) จะนำค่า Bravo ไปออกจอภาพ

สถานะของพอยน์เตอร์

รูปแบบของการใช้งานพอยน์เตอร์กับตัวแปรคิดว่ามีอยู่เพียงเท่านี้ จะเห็นว่าไม่มีความจำเป็นต้องใช้พอยน์เตอร์ เพราะสามารถใช้ตัวแปรได้โดยตรงอยู่แล้ว จึงขอนำเรื่องกับข้อมูลอิสระมากล่าวถึง แต่จะขอแสดงสถานะของพอยน์เตอร์ในโปรแกรมก่อนคือ

ตามโปรแกรมตัวอย่างในรูปที่ 3 เมื่อเรากำหนดตัวแปรพอยน์เตอร์ iptr จะมีค่าเถื่อน ๆ อยู่ในตัวมัน ทำให้มันชี้ไปที่แอดเดรสใดก็ได้ หากเราให้ค่าข้อมูลเช่นที่ให้ค่า 123 ที่ต้นโปรแกรมจะเกิดปัญหาทันที เพราะจะไปเปลี่ยนค่าที่แอดเดรสนั้น ซึ่งอาจจะเป็นในโปรแกรมระบบจัดการก็ได้เป็น 123 สถานะเช่นนี้ถือว่าไม่ปลอดภัย

```
VAR   iptr : ^Integer;
      num  :  Integer;
BEGIN
  iptr := Addr(num);
  iptr^ := 123;
  New(iptr);
  iptr^ := 345;
  Dispose(iptr);
  iptr := nil;
END.
```

รูปที่ 3 แสดงสถานะของพอยน์เตอร์ในโปรแกรม

เมื่อเราให้ค่าแอดเดรสของตัวแปร num แก่มันเท่ากับเป็นการกำหนดหน่วยความจำแก่มัน สถานะเช่นนี้ถือว่าปลอดภัย เพราะเมื่อเราให้ค่าคงตัว 123 แก่มันจะเป็นการให้ค่านี้แก่ตัวแปร num ด้วย

ต่อไปเมื่อใช้คำสั่ง New (iptr) จะเป็นการกำหนดหน่วยความจำในฮีพ (Heap) ให้แก่มัน (ฮีพคืออะไรขอพักไว้ก่อนครับ) สถานะนี้เป็นสถานะที่ปลอดภัยและเป็นการใช้งานพอยน์เตอร์โดยทั่วไป ซึ่งได้ให้ค่า 345 ไว้แก่หน่วยความจำนี้

แต่ยังไม่ได้ใช้งานอะไรผมก็ยกเลิกหน่วยความจำในฮีพด้วยคำสั่ง Dispost (iptr) ตรงนี้ครับที่สงสัยกันมากพอสมควร เพราะ iptr ยังคงชี้หน่วยความจำหน่วยนี้อยู่ ค่า 345 ก็ยังคงมีอยู่ แต่โปรแกรมจัดการฮีพ (Heap manager) ถือว่ายกเลิกแล้ว หากนำไปใช้งานอื่นค่าก็เปลี่ยนไป จึงถือว่าเป็นสถานะที่ไม่แน่นอน และจะ

```
PROGRAM   PointerTwo(output);
TYPE      nameT = String[30];
VAR       aname, bname : nameT;
          ptr1, ptr2   : ^nameT;
          tem          : Pointer;

  procedure  print(VAR name : nameT);
  begin  writeln(name)  end;

BEGIN
  ptr1  :=  Addr(aname);  ptr2  :=  @bname;
  aname := 'Alpha';       bname := 'Bravo';
  WriteLn(ptr1^,'  ',ptr2^);
  tem := ptr1;  ptr1 := ptr2;  ptr2 := tem;
  WriteLn(ptr1^,'  ',ptr2^);

  Print(ptr1^);
writeln
END.
```

รูปที่ 2 แสดงการใช้งานพอยน์เตอร์กับตัวแปรสแตติกส์

ไม่ใช้ตัวแปรพอยน์เตอร์ต่อไปจนกว่าจะกำหนดหน่วยความจำให้แก่มันใหม่

คำสั่งสุดท้ายคือ iptr: = nil เป็นการยกเลิกไม่ให้ iptr ชี้ไปที่ใด แต่ความจริงเป็นการให้ค่าศูนย์ คือให้ชี้ไปที่แอดเดรส 0000 : 0000 จึงเป็นอันตรายอีกเช่นกัน ที่กำหนดไว้ก็เพื่อจะตรวจสอบว่าพอยน์เตอร์ใดว่าง ซึ่งต้องขอนำไปกล่าวในเรื่องพอยน์เตอร์กับข้อมูลแบบโครงสร้าง

ก็พอสรุปได้ว่า **อย่าใช้งานพอยน์เตอร์ถ้ายังไม่ได้กำหนดหน่วยความจำให้แก่มัน**

เรื่องของหน้าต่าง

ดังได้กล่าวแล้วว่าการใช้พอยน์เตอร์กับข้อมูลอิสระ ส่วนมากใช้ในโปรแกรมประเภทเวิร์ดโปรเซสเซอร์ แต่การยกตัวอย่างในเรื่องนี้คงจะไม่สนุกนัก จึงขอยกตัวอย่างเป็นการสร้างหน้าต่าง (ถ้าคิดว่าเป็นเรื่องเก่าก็ขอให้ติดตามไปก่อนครับ)

รูปแบบ

เมื่อกล่าวถึงการสร้างหน้าต่างก็ต้องพิจารณาถึงรูปแบบของหน้าต่างก่อน เพราะสร้างได้หลายรูปแบบมาก กล่าวคือ

- จะให้ซ้อนทับกันได้หรือไม่ได้ จะขอเลือกแบบของการสร้างหน้าต่างในแบบซ้อนทับกันได้หลาย ๆ หน้าต่างมากล่าวถึง เพราะทำได้ยากกว่า และเมื่อทราบหลักการแล้ว สามารถนำไปทำในแบบไม่ซ้อนทับกันได้ด้วย
- จะให้เลื่อนหน้าต่างไปมาตามความต้องการของผู้ใช้ได้หรือไม่ จะให้เลื่อนได้เพราะสนุกดีครับ และไม่ยากด้วย
- การใช้หน้าต่างในแต่ละครั้งจะดำเนินการอย่างไร ก็ขอเรียนให้ทราบว่าการใช้หน้าต่างหลาย ๆ หน้าต่าง เทียบได้กับการวางแผ่นใสซ้อนกัน แผ่นใสหนึ่ง ๆ จะมีช่องหน้าต่างอยู่ 1 ช่อง การใช้หน้าต่างจึงทำได้ 3 แบบคือ

– วางแผ่นใสทีละแผ่นแล้วยกออกวางแผ่นอื่นแทนเมื่อต้องการ

– วางแผ่นใสซ้อนทับกันตามที่ต้องการไปเรื่อย ๆ เช่น ต้องการใช้หน้าต่างหนึ่งก็วางแผ่นที่ 1 เมื่อต้องการใช้หน้าต่างสามก็วางแผ่นที่ 3 ทับ ต้องการใช้หน้าต่างสี่ก็วางแผ่นที่ 4 ทับต่อไปอีก ต่อไปหากต้องการใช้หน้าต่างหนึ่งก็หยิบแผ่นที่ 4 กับแผ่นที่ 3 ออก

– วางแผ่นใสของหน้าต่างทั้งหมดตามลำดับแต่เริ่มต้น เช่นมี 5 หน้าต่างก็จะวางเป็น 1,2,3,4,5 โดยมีแผ่นที่ 5 อยู่บนสุด เมื่อผู้ใช้เลือกหน้าต่างใดก็จะหมุนวนนำหน้าต่างนั้นมาอยู่บน เช่นเลือกหน้าต่างสองก็จะหมุนจนเป็น 3,4,5,1,2

จะเลือกแบบหมุนวนครับ เพราะสนุกดีและเมื่อทราบเทคนิคแล้วจะดำเนินการแบบใดก็ได้ นอกจากนี้จะนำหน้าต่างอื่นที่ไม่ได้อยู่ในพวกหมุนวนมาซ้อนทับอีกก็ได้

ขั้นตอนดำเนินการ

ดร.เวียร์ต กล่าวว่า Algorithms + Data Structures = Programs คือ ขั้นตอนวิธี + โครงสร้าง = โปรแกรม แต่ผมขอกล่าวเสริมว่าต้องรู้ความมุ่งหมายที่แน่นอนด้วย จึงจะเขียนโปรแกรมได้ ซึ่งความมุ่งหมายในเรื่องนี้ได้กำหนดไว้แน่นอนแล้วในเรื่องรูปแบบ ขั้นต่อไปคือหาขั้นตอนวิธี ซึ่งอาจจะมีผู้กำหนดไว้แล้ว หรือได้จากการวิเคราะห์งาน หรือได้จากความรู้เดิม หรือประสบการณ์

ซึ่งเรื่องของการใช้หน้าต่างมีเรื่องที่เราพอมองเห็นกันอย่างง่าย ๆ ว่าจะแบ่งออกเป็น 3 ขั้นตอนคือ เปิดหน้าต่าง ใช้งานหน้าต่าง และปิดหน้าต่าง ซึ่งในแต่ละขั้นตอนยังมีขั้นตอนย่อยที่จะต้องพิจารณาหรือวิเคราะห์อีก และเป็นเรื่องใหญ่พอสมควร จึงขอยกเป็นหัวข้อรอง

แต่เพื่อเป็นพื้นฐานในการพิจารณาขอเรียนให้ทราบก่อนว่าในเทอร์โบปาสคาลมีคำสั่ง Window (x1, y1, x2, y2) อยู่ เพื่อเปิดหน้าต่างตามโคออร์ดิเนตที่ระบุ การเปิดหน้าต่างหมายถึงว่าการใช้หน้าต่างนั้นแทนจอภาพทั้งจอ เช่นใช้คำสั่งว่า ClrScr จะเป็นการลบจอภาพในช่องหน้าต่างนี้เท่านั้น คำสั่ง GotoXY ก็จะเลื่อนเคอร์เซอร์ไปตรงจุดโคออร์ดิเนตที่สัมพันธ์กับหน้าต่างนี้เท่านั้น เพื่อเสริมความเข้าใจ หากมีโอกาสขอให้ทดลองรันโปรแกรมในรูปที่ 4 จะได้ผลการรันดังรูปที่ 5

เปิดหน้าต่าง

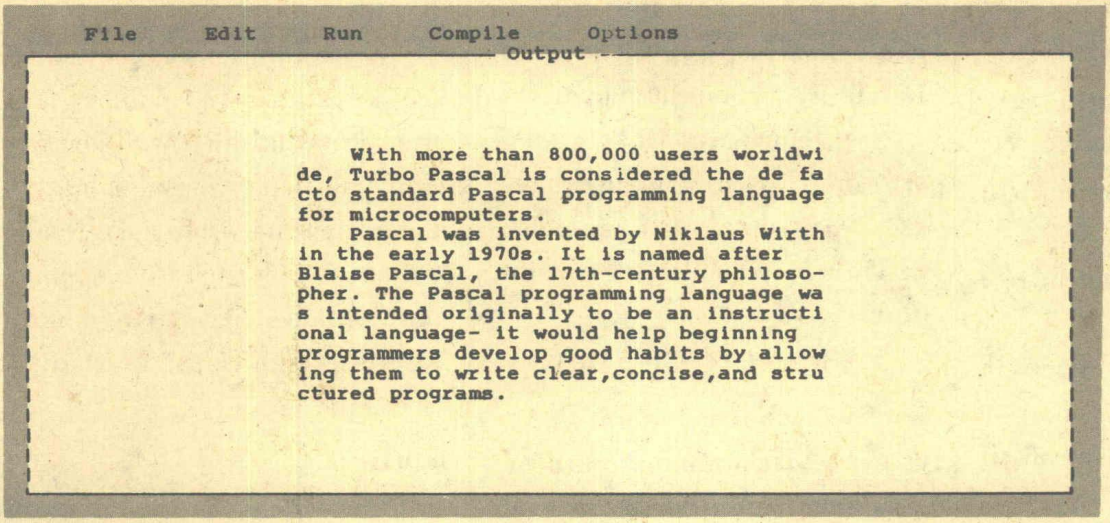
ในการเปิดหน้าต่างเนื่องจากคำสั่ง Window ยังขาดการเตรียมการ ซึ่งเราจะต้องดำเนินการเองตามลำดับดังนี้

- **เก็บข้อมูลบนจอภาพเดิม**

ซึ่งมีข้อพิจารณาว่าจะเก็บเฉพาะตรงช่องหน้าต่างใหม่หรือทั้งจอภาพ การเก็บทั้งจอภาพมีความอ่อนตัวในการใช้งานดี แต่เปลืองเนื้อที่ในหน่วยความจำมากกว่า โดยปกติจะใช้กับหน้าต่างขนาดใหญ่ โดยเฉพาะที่ต้องการให้เลื่อนหน้าต่างไปมาตามความต้องการของผู้ใช้ได้ จึงจะเลือก

```
PROGRAM  Pointer4 (input,output);
USES     crt;
VAR      astr : String;
BEGIN
  ClrScr;
  Window(21,5,60,20);
  REPEAT
    ReadLn(astr)
  UNTIL  astr = '' ;
  Window(1,1,80,25);
END.
```

รูปที่ 4 โปรแกรมสร้างหน้าต่างอย่างง่าย



รูปที่ 5 ผลการรันโปรแกรม Pointer 4 ข้อความที่เห็นเป็นข้อความที่ป้อนเพื่อแสดงถึงการตกบรรทัดในช่องหน้าต่าง

เก็บในแบบนี้

จะเก็บไว้ที่ใด สามารถทำได้ 3 แบบ คือ สำหรับการ์ดเฮอร์คิวลิสสามารถเก็บในหน่วยความจำจอภาพที่เหลือประมาณ 60 กิโลไบต์ (ซึ่งได้กล่าวแล้วในไมโครฯ ฉบับที่ 27) หรือเก็บในตัวแปร หรือเก็บในหน่วยความจำฮีพ คือข้อมูลอิสระ ขอเลือกเป็นฮีพครับ เพราะเรื่องนี้จะกล่าวถึงพอยน์เตอร์ ซึ่งข้อดีข้อเสียจะกล่าวภายหลัง

ตำบลที่ของเคอร์เซอร์เดิม จะต้องเก็บไว้ด้วยเพื่อนำมาคืนให้เมื่อปิดหน้าต่าง

● ตีกรอบ

ที่ต้องตีกรอบก็เพื่อให้เห็นช่องหน้าต่างเด่นชัด ขนาดของกรอบต้องใหญ่กว่าขนาดของหน้าต่างอยู่ 1 ช่องอักขระทุกด้าน ในการกำหนดขนาดของหน้าต่างจะให้กำหนดขนาดของกรอบหรือกำหนดขนาดของหน้าต่าง จะให้กำหนดขนาดของกรอบเพราะเขียนโปรแกรมง่ายกว่า ดังนั้นหากต้องการขนาดของหน้าต่างแท้ ๆ เช่น 20×10 จะต้องกำหนดเป็น 22×12

การตีกรอบจะตีทุกครั้งที่เปิดใช้หน้าต่างหรือตีครั้งแรกครั้งเดียวเมื่อเตรียมหน้าต่าง จากการเลือกวิธีเก็บข้อมูลบนจอภาพเดิมทั้งจอภาพ เปิดโอกาสให้ตีกรอบในครั้งแรกครั้งเดียว ไม่ต้องตีกรอบบ่อย ๆ แต่จะต้องเก็บขนาดของกรอบด้วย

● ลอกข้อมูลของตนเอง

เมื่อใช้งานหน้าต่างแล้ว จะมีข้อมูลค้างอยู่ซึ่งต้องเก็บไว้ และนำออกแสดงเมื่อกลับมาเปิดหน้าต่างเดิมในครั้งต่อไป จะดำเนินการอย่างไรขอนำไปกล่าวในหัวข้อปิดหน้าต่าง

● เปิดใช้หน้าต่าง

ขั้นตอนนี้ไม่มีสิ่งใดต้องพิจารณาการดำเนินการ มีแต่เพียงเปิดใช้หน้าต่างจริง ๆ ด้วยคำสั่ง Window และให้เคอร์เซอร์ไปอยู่ ณ ตำแหน่งที่กำหนด หน้าต่างก็พร้อมที่จะใช้งานแล้ว

ใช้งานหน้าต่าง

การทำงานในแต่ละหน้าต่าง จะเป็นไปตามโปรแกรมที่เขียนขึ้นต่อไป ในการทดลองจะให้อ่านสตริงไปจนกว่าจะกดคีย์ ENTER ที่ต้นสตริงเท่านั้น

ปิดหน้าต่าง

เก็บจอภาพปัจจุบันคือของหน้าต่างบนสุดที่กำลังเปิดใช้อยู่ รวมทั้งตำแหน่งเคอร์เซอร์ด้วย (ข้อพิจารณาเช่นเดียวกับเมื่อเปิดหน้าต่าง)

นำจอภาพของหน้าต่างล่าง (ที่เก็บไว้) มาแทนที่ และเปิดใช้หน้าต่างนี้เป็นหน้าต่างปัจจุบันแทน

จะเห็นว่าในการปิดหน้าต่างนี้ ข้อมูลในช่องหน้าต่างล่างจะถูกลอกโดยอัตโนมัติ เมื่อนำจอภาพของหน้าต่างล่างมาแทนที่ แต่ในการเปิดหน้าต่างจะทำเช่นนี้ไม่ได้ จอภาพของหน้าต่างล่างจะถูกทับ ดังนั้น ในการเปิดหน้าต่างจะต้องลอกเฉพาะภาพในช่องหน้าต่างใหม่มาลงบนจอภาพ

สรุปการดำเนินการ

ผมขอสรุปการดำเนินการโดยจะยกตัวอย่าง การเปิดใช้หน้าต่าง 1 เพียงหน้าต่างเดียว ซึ่งจะเปิดทับบนจอภาพ และจะเรียกจอภาพว่าหน้าต่าง 0 โดยที่หน้าต่างหนึ่ง ๆ อยู่บนหน่วยความจำขนาดเท่าจอภาพ แต่การใช้งานเทียบเท่าแผ่นใส จึงจะขอเรียกว่าแผ่นใส เพื่อแยกจากจอภาพ จะได้การดำเนินการตามลำดับดังนี้

● เตรียมหน้าต่าง 1

- ลบแผ่นใสหน้าต่าง 1
- ตีกรอบหน้าต่างของหน้าต่าง 1
- กำหนดตำแหน่งเคอร์เซอร์ไว้ที่มุมบนซ้ายของช่องหน้าต่าง 1

● เปิดหน้าต่าง

- เก็บภาพบนจอภาพลงแผ่นใสของหน้าต่าง 0
- ลอกภาพเฉพาะในช่องหน้าต่าง 1 รวมกรอบหน้าต่างจากแผ่นใสของหน้าต่าง 1 ลงจอภาพ
- เปิดใช้หน้าต่าง 1 ด้วยคำสั่ง Window

– ให้เคอร์เซอร์อยู่ ณ ตำแหน่งที่กำหนด

- **ใช้งานหน้าต่าง** (ตามโปรแกรม)
- **ปิดหน้าต่าง**

– เก็บภาพบนจอภาพลงแผ่นใสของหน้าต่าง 1

– นำภาพจากแผ่นใสของหน้าต่าง 0 ออกจอภาพ

– เปิดใช้หน้าต่าง 0 ด้วยคำสั่ง Window

– ให้เคอร์เซอร์อยู่ ณ ตำแหน่งที่เก็บไว้

ทั้งหมดนี้คือขั้นตอนวิธีของการดำเนินการกับหน้าต่างขั้นพื้นฐาน สำหรับการเลื่อนหน้าต่าง และการหมุนวนหน้าต่างขอพักไว้ก่อนครับ

โครงสร้างข้อมูล

ตามที่ ดร.เวียร์ต กล่าว เมื่อเราได้ขั้นตอนวิธีแล้ว ก็ต้องกำหนดแบบของข้อมูลที่เหมาะสม ซึ่งแบบของข้อมูลที่จะนำมาใช้นี้เป็นแบบพื้นฐานเท่านั้น (Primitive data structure) ดังจะกล่าวต่อไปนี้

แต่มีเรื่องที่จะต้องทราบก่อนครับ เพราะต้องอาศัยความรู้ทางด้านฮาร์ดแวร์ด้วยคือ หน่วยความจำจอภาพที่ผมใช้คำว่าจอภาพตลอดมานั้น เป็นหน่วยความจำขนาด 4000 ไบต์อยู่ที่แอดเดรส $B000 : 0000 (เป็นแอดเดรสเริ่มต้น)

- **เรคอร์ด**

จากการวิเคราะห์งานที่ผ่านมาจะเห็นว่ามีข้อมูลที่เป็นตัวเอกอยู่ 3 เรื่องคือ จอภาพกับแผ่นใส จุดโคออร์ดิเนตของกรอบ และตำบลที่ของเคอร์เซอร์ ซึ่งจะต้องเก็บไว้เป็นของแต่ละหน้าต่างไป จึงควรรวมเป็นข้อมูลชุดเดียวกันสำหรับแต่ละหน้าต่าง แบบของข้อมูลที่เหมาะสมจึงควรเป็นเรคอร์ด ซึ่งมีเรื่องจะต้องพิจารณาในแต่ละฟิลด์อีก คือ

จอภาพโดยปกติจะอ้างถึงจอภาพในแบบ 25 แถว × 80 คอลัมน์ คอลัมน์ละ 2 ไบต์ แบบของข้อมูลที่เหมาะสมสำหรับฟิลด์นี้จึงควรเป็น ดังรูปที่ 6 แบบใดแบบหนึ่ง จะเลือกเอาแบบแรก

จุดโคออร์ดิเนตของกรอบ ปกติจะอ้างเป็น x1, y1 สำหรับมุมบนซ้าย และ x2, y2 สำหรับมุมล่างขวา แต่เพื่อความสะดวกในการเขียนโปรแกรมจะกำหนดเป็นอะเรย์คือ

ARRAY [1..4] OF Byte

โดยที่ในแต่ละช่องจะเก็บเป็นค่า x1, x2, y1 และ y2 เรียงลำดับกันจากช่องแรก

สำหรับตำบลที่ของเคอร์เซอร์จะเก็บเป็นไบต์ตรง ๆ สรุปรวมทั้งหมดที่กล่าวมานี้จะได้เป็นแบบข้อมูล ScreenT ในรูป I1 ที่ให้เป็น I เพราะจะจัดให้โปรแกรมส่วนนี้เป็น Include file ที่ต้องแยก edgeT ไว้ต่างหากเพราะต้องนำไปใช้อ้างอิงอย่างอื่นอีก และต้องนำไปกำหนดในโปรแกรมหลัก ขอให้พลิกไปดูในรูป A ครับ

อะเรย์

การดำเนินการกับข้อมูลแบบเดียวกันเป็นจำนวนมาก ในกรณีนี้คือหน้าต่าง หากกำหนดตัวแปรอิสระแต่ละตัวจะต้องเขียนโปรแกรมยาว หากกำหนดเป็นอะเรย์จะสะดวกกว่า และดังกล่าวแล้วว่าจะเก็บข้อมูลของหน้าต่างในฮีพแบบของข้อมูลที่เหมาะสมคือ อะเรย์ของพอยน์เตอร์ ดังที่กำหนดให้แก่ตัวแปร wnd ในรูป I1 เช่นกัน

สแต็ก

– สแต็กก็คืออะเรย์นั่นเอง แต่เรามองในแง่ของการใช้งานว่าเป็นสแต็ก คือการวางของ เช่น จานซ้อนกัน จึงเหมาะสำหรับหน้าต่าง เพื่อที่จะติดตามว่าขณะนี้หน้าต่างใดอยู่ชั้นบนสุดซ้อนอยู่บนหน้าต่างใดบ้าง จึงได้กำหนดตัวแปร stack ทำหน้าที่นี้

สำหรับการกำหนดตัวแปรอื่น ๆ เป็นไปตามความต้องการใช้งานคือ screen

```
TYPE
   ScreenT = RECORD
      scr   : ARRAY[1..25,1..80] OF Word;
      curX, curY : ARRAY[1..4] OF Byte;
            END;

   ScreenT = RECORD
      scr   : ARRAY[1..25,1..80] OF RECORD
                                      ch   : Char;
                                      attr : Byte;
                                      END;
      curX, curY : ARRAY[1..4] OF Byte;
```

รูปที่ 6 แบบข้อมูลหน้าต่าง 2 แบบ

```
(*====================  WINDOW.ICL ========================*)

TYPE
        ScreenT = RECORD
          scr    : ARRAY[1..25,1..80] OF Word;
          curX, curY : Byte;
          edge   : edgeT;
                 END;
        WndPtr = ^ScreenT;
VAR     screen : ScreenT ABSOLUTE $B000:$0000;
        wnd : ARRAY[0..MAXWNDW] OF WndPtr;
        actwnd, ptrscrn, temw : WndPtr;
        stack : ARRAY[0..MAXSTCK] OF Byte;
        sp : Byte;
        astr : String;
        _cursX, _cursY : Byte;
```

I1

ให้เป็นตัวแปรแบบ ScreenT และให้อยู่ที่หน่วยความจำที่เป็นจอภาพจริง ๆ เพื่อจะดำเนินการกับจอภาพได้โดยตรง

ตัวแปร actwnd (อ่านว่า actual window) กำหนดเพื่อจะใช้ชี้หน้าต่างที่กำลังเปิดใช้งานอยู่ ความจริงผมกำหนดเพลินไปเหมือนกันเพราะ actwnd ก็คือ wnd [stack[sp]] นั่นเอง

ส่วนตัวแปร ptrscrn กำหนดขึ้นเพื่อจะถ่ายเทข้อมูลจากจอภาพคือ screen ไปให้แก่หน้าต่างคือ wnd[x] ได้สะดวกขึ้นเท่านั้น ความจริงแล้วกำหนด screen หรือ ptrscrn ตัวใดตัวหนึ่งก็ได้ เช่นให้ค่า ptrscrn เป็น

Addr (Cseq ($B000); ofs ($0000))

แต่ก็ยังดีกว่าหนังสือต่างประเทศเล่มหนึ่ง ที่ New (ptrscrn) ไว้ในฮีพแล้วมากำหนดแอดเดรสของจอภาพให้แก่ ptrscrn อีก ปล่อยหน่วยความจำในฮีพไว้โดยไม่มีทางจะติดต่อกับใครอีก สำหรับของผมก็เพียงรกตาแต่ไม่เสียหายอะไร ตัวแปรอื่น ๆ นอกนั้นยังไม่ถึงบทบาทครับ

พอยน์เตอร์กับหน้าต่าง

ความจริงบทความนี้เป็นเรื่องของพอยน์เตอร์ แต่เขียนไปเขียนมากลายเป็นเรื่องของหน้าต่างไป บทบาทของพอยน์เตอร์จะอยู่ในโปรแกรมครับแทบทุกบรรทัดเลย

จากการกำหนดตัวแปรที่ผ่านมาบางท่านอาจสงสัยว่าในโครงงานนี้จำเป็นหรือไม่ที่จะต้องใช้พอยน์เตอร์ คำตอบก็คือขึ้นอยู่กับลักษณะงานที่จะใช้หน้าต่างอีกทอดหนึ่ง กล่าวคือ ในเทอร์โบปาสคาลจัดเนื้อที่สำหรับข้อมูลได้เพียง 64 กิโลไบต์ แต่มีเนื้อที่เหลืออยู่ในฮีพอีกหลายร้อยกิโลไบต์ ผมชอบคำพูดในหนังสือฝรั่งเล่มหนึ่งที่ว่า "The memory is out there" แต่ไม่ค่อยใช้กัน

จึงขึ้นอยู่ว่าในโปรแกรมจะเก็บข้อมูลทั่วไปไว้ที่ใด ถ้าเก็บในดาต้าเซกเมนต์คือ 64 กิโลไบต์ หน้าต่างก็ควรจะไปอยู่ในฮีพซึ่งต้องใช้พอยน์เตอร์ ถ้าเก็บข้อมูลในฮีพหน้าต่างก็ควรอยู่ในดาต้าเซกเมนต์ ซึ่งไม่ต้องใช้พอยน์เตอร์สำหรับหน้าต่าง ผมใช้คำว่าควรเพราะจะให้อยู่ด้วยกันก็ได้ไม่มีข้อขัดข้องประการใด

สร้างเครื่องมือไว้ใช้งาน

ในการเขียนโปรแกรมจะเขียนจากบนลงล่าง หรือจะเขียนจากล่างขึ้นบนก็ได้ แต่เนื่องจากโครงงานนี้ยาวเกินกว่าที่จะจบในฉบับเดียว ผมจึงแบ่งโครงงานนี้ออกเป็น 3 ช่วงคือ

– ช่วงแรกให้สร้างหน้าต่างได้ รวมทั้งให้ผู้ใช้กำหนดขนาดของหน้าต่างได้ด้วย

– ช่วงที่สองให้ผู้ใช้สามารถเลื่อนหน้าต่างไปมาได้

– ช่วงที่สามให้ผู้ใช้สามารถเลือกหน้าต่างโดยวิธีวนรอบได้

สำหรับช่วงที่สองที่สามคงต้องเป็นฉบับหน้าครับ เมื่อเป็นเช่นนี้การเขียนโปรแกรมจะเป็นในลักษณะจากล่างขึ้นบน คือ จะเขียนโปรแกรมย่อยในลักษณะสร้างเครื่องมือไว้ใช้งานคือ ต้องให้อ่อนตัวพอสมควร โดยจะเริ่มตามลำดับที่ได้วิเคราะห์งานไว้แล้ว และจะใช้ชื่อหัวข้อเป็นชื่อโปรแกรมย่อยเลย คือ

SetUpWindow

คือ การเตรียมหน้าต่างนั่นเองอยู่ในรูป I3 เริ่มด้วยการเตรียมหน้าต่างของจอภาพ คือหน้าต่าง 0 คือ wnd[0] โดยลบจอภาพให้ค่าเคอร์เซอร์ของจอภาพเป็น 1,1 ให้ค่ากรอบหน้าต่างเต็มจอภาพ (ค่ากรอบหน้าต่าง wndsize กำหนดไว้ในโปรแกรมหลักรูป A) เสร็จแล้วสร้างแผ่นใสของหน้าต่าง wnd[0] ขึ้นด้วยคำสั่ง New (wnd[0]) แล้วให้ข้อมูลทั้งหมดของจอภาพแก่หน้าต่าง wnd[0] ให้ค่า actwnd และ stack กับ sp (อ่านว่า stack pointer) ให้ตรงกับความจริง

ท่อนต่อไปเป็นการเตรียมหน้าต่างที่เหลือ ซึ่งให้พิมพ์ออกมาด้วยว่าเตรียมถึงหน้าต่างไหนแล้ว ไม่เช่นนั้นจะดูเหงา ๆ ไป ขั้นตอนต่อไปคล้าย ๆ กับของจอภาพ มีที่สังเกตว่าได้ให้ข้อมูลของจอภาพแก่ทุกหน้าต่างเหมือนกัน ดังนั้นทุกหน้าต่างจะถูกเคลียร์ และมีตำแหน่งของเคอร์เซอร์เป็น 1,1 เหมือนกัน แต่จะเป็น 1,1 ของหน้าต่างนั้น ๆ จากนั้นได้ให้ค่ากรอบหน้าต่างของแต่ละหน้าต่างทับ เป็นของใครของมัน เสร็จแล้วให้ตีกรอบ

DrawFrame

คือการตีกรอบอยู่ในรูป I2 ขอให้สังเกตส่วนหัวครับ ได้กำหนดไว้เป็น

I3

```
PROCEDURE  SetUpWindow;
VAR   I, J : Integer;
BEGIN
  ptrscrn := @screen;
  ClrScr;  screen.curX := 1;  screen.curY := 1;
  screen.edge := wndSize[0];
  New(wnd[0]);   wnd[0]^ := ptrscrn^ ;  actwnd := wnd[0];
  stack[0] := 0;  sp := 0;

  FOR  I := 1 TO MAXWNDW  DO
    BEGIN
    WriteLn('Window ',I);
    New(wnd[I]);  wnd[I]^ := ptrscrn^;
    wnd[I]^.edge := wndsize[I];
    DrawFrame(wnd[I],wnd[I]);
    END;
END;
```

modl กับ dstn ซึ่งความจริงแต่ละหน้าต่างก็มีค่าโคออร์ดิเนตของตนอยู่แล้ว จะตีกรอบหน้าต่างใดก็บ่งได้ตรงตัว คือ กำหนดตัวแปรเพียงตัวเดียวก็พอ แต่ดังได้กล่าวแล้วว่าจะเขียนโปรแกรมย่อยไว้เป็นเครื่องมือเพื่อจะใช้กรอบของหน้าต่างหนึ่งไปตีบนอีกหน้าต่างหนึ่ง เช่นในกรณีจะตีกรอบตรงกับจอภาพ

สำหรับการผ่านค่าตัวแปรพอยน์–เตอร์ต้องผ่านในแบบ pass by reference คือใช้คำว่า VAR เสมอ (เช่นเดียวกับไฟล์) การทำงานของโปรแกรมนี้ก็เพียงแต่ตีกรอบ โดยตีกรอบด้านบน–ล่างก่อน แล้วตีกรอบด้านข้าง แล้วจึงให้ภาพมุมทั้ง 4 ด้าน แบบกรอบได้ให้ไว้เป็นค่าคงตัวเปลี่ยนได้ตามใจชอบ มีอีกเรื่องหนึ่งคือผมควรจะให้ชื่อหน้าต่างด้วย แต่ไม่ได้ให้เพราะเกรงว่าโปรแกรมจะยาว

OpenWindow

คือการเปิดหน้าต่างอยู่ในรูป I5 ด้วยการเรียกโปรแกรมย่อย SaveScreen เสร็จแล้วให้ actwnd ชี้ไปที่หน้าต่างใหม่ และเก็บหมายเลขหน้าต่างลงสแต็ก จะเห็นว่า sp จะชี้ที่หมายเลขหน้าต่างที่อยู่บนสุดของจอภาพเสมอ หลังจากนั้นให้ลอกภาพเฉพาะตรงช่องหน้าต่างไปออกจอภาพด้วย CopyWindow แล้วเปิดใช้หน้าต่างด้วย ActiveWindow จะเห็นว่าเขียนโปรแกรมตรงกับที่วิเคราะห์งานไว้ ต่างกันตรงที่ฝากงานกำหนดเคอร์เซอร์ไปให้แก่ ActiveWindow ด้วย

SaveScreen

เก็บภาพบนจอภาพลงแผ่นใสของหน้าต่างที่ระบุโดยเก็บตำแหน่งของเคอร์–เซอร์ด้วย (อยู่ในรูป I4)

CopyWindow

ลอกภาพตรงช่องหน้าต่างจากแผ่นใสของหน้าต่างใหม่ลงจอภาพ ซึ่งจะต้องให้ค่ากรอบและตำแหน่งของเคอร์เซอร์ที่เก็บไว้ด้วย (อยู่ในรูป I4)

I2

```
PROCEDURE  DrawFrame(VAR modl,dstn : WndPtr);
CONST  UPL = $07DA; UPR = $07BF; LOL = $07C0; LOR = $07D9;
       VER = $07C4; HOR = $07B3;
VAR    x1,x2,y1,y2, row, col : Byte;
BEGIN
  x1 := modl^.edge[1];  x2 := modl^.edge[2];
  y1 := modl^.edge[3];  y2 := modl^.edge[4];
  WITH  dstn^  DO
    BEGIN
    FOR  col := x1 TO x2  DO
      BEGIN  scr[y1,col] := VER;  scr[y2,col] := VER  END;
    FOR  row := y1 TO y2  DO
      BEGIN  scr[row,x1] := HOR;  scr[row,x2] := HOR  END;
    scr[y1,x1] := UPL;  scr[y1,x2] := UPR;
    scr[y2,x1] := LOL;  scr[y2,x2] := LOR;
    END;
END;
```

I5

```
PROCEDURE  OpenWindow(n : Byte);
BEGIN
  SaveScreen(actwnd);
  actwnd := wnd[n];  sp := sp + 1;  stack[sp] := n;
  CopyWindow(actwnd, ptrscrn);
  ActiveWindow(actwnd);
END;

PROCEDURE  CloseWindow;
VAR  num : Byte;
BEGIN
  SaveScreen(actwnd);
  IF  sp > 0  THEN
    BEGIN
    sp := sp - 1;  num := stack[sp];
    actwnd := wnd[num];  ptrscrn^ := actwnd^;
    ActiveWindow(actwnd);
    END;
END;
```

I4

```
PROCEDURE  SaveScreen(VAR dstn : WndPtr);
BEGIN
  screen.curX := WhereX;  screen.curY := WhereY;
  dstn^ := ptrscrn^
END;

PROCEDURE  CopyWindow(VAR srce, dstn : WndPtr);
VAR    row,col : Byte;
BEGIN
  FOR  row := srce^.edge[3] TO srce^.edge[4]  DO
    FOR  col := srce^.edge[1] TO srce^.edge[2]  DO
      dstn^.scr[row,col] := srce^.scr[row,col];
  dstn^.edge := srce^.edge;
  dstn^.curX := srce^.curX;  dstn^.curY := srce^.curY;
END;

PROCEDURE  ActiveWindow(VAR w : WndPtr);
VAR  x1,x2,y1,y2 : Byte;
BEGIN
  WITH  w^  DO
    BEGIN  x1 := edge[1] + 1;   x2 := edge[2] - 1;
           y1 := edge[3] + 1;   y2 := edge[4] - 1  END;
  Window(x1,y1,x2,y2);
  WITH  w^  DO  GotoXY(curX,curY);
END;
```

ActiveWindow

เปิดใช้หน้าต่างใหม่ โดยต้องคำนวณลดขนาดของหน้าต่างแท้ ๆ ให้เล็กกว่ากรอบ และวางเคอร์เซอร์ให้อยู่ ณ ตำแหน่งที่เก็บไว้ด้วย (อยู่ในรูป I4)

CloseWindow

ปิดหน้าต่างที่เปิดไว้โดย เก็บภาพบนจอภาพขณะนั้นลงแผ่นใสของหน้าต่างที่กำลังเปิดอยู่ พร้อมด้วยตำบลที่ของเคอร์เซอร์ด้วย SaveScreen

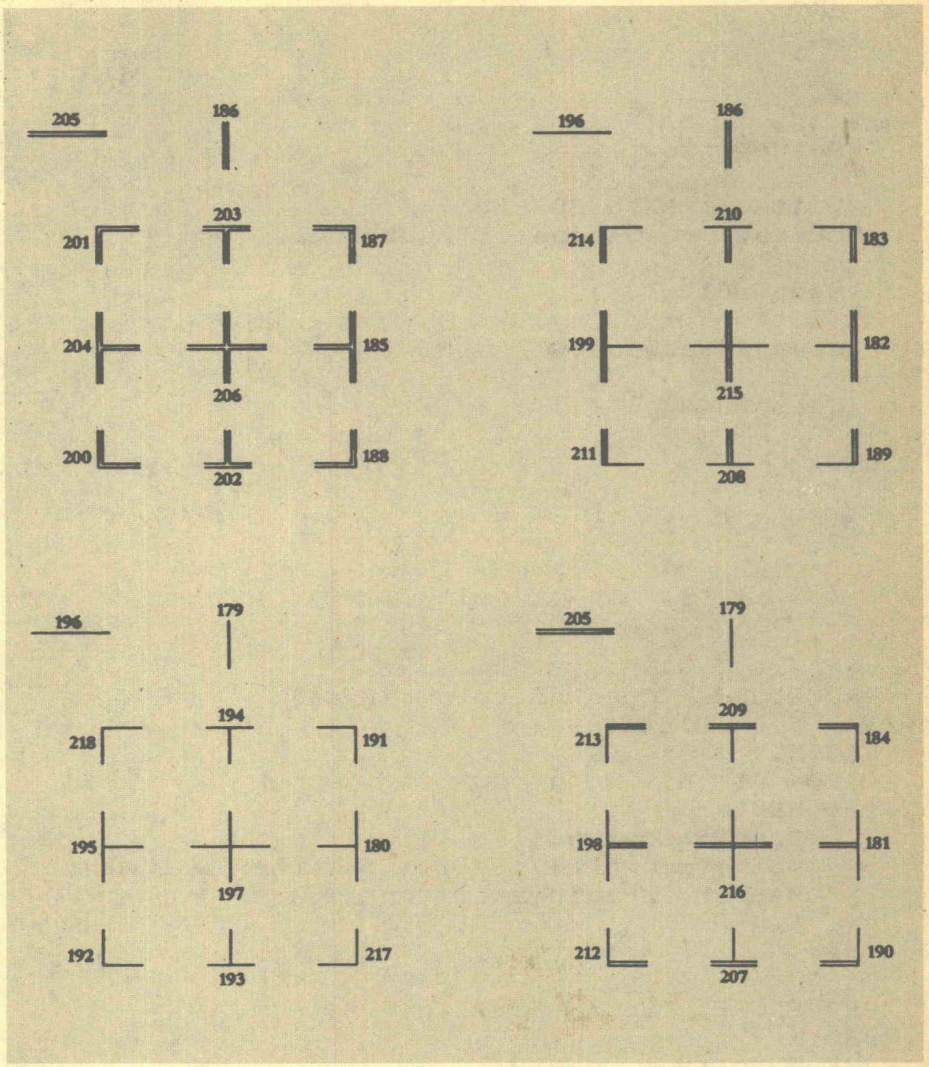
เสร็จแล้วให้ actwnd ชี้ที่หน้าต่างซึ่งอยู่ชั้นล่างลงไป 1 ชั้น เพื่อจะย้ายข้อมูลทั้งหมดของแผ่นใสนี้มาให้จอภาพด้วยคำสั่ง ptrscrn^ : = actwnd^(ข้อมูลที่ย้ายมาคือ ภาพของจอภาพ ตำแหน่งเคอร์เซอร์ และค่าโคออร์ดิเนตของกรอบ) เสร็จแล้วเปิดใช้หน้าต่างนี้ด้วย Active-Window

โปรแกรมย่อยสำหรับใช้งานในช่วงแรกนี้มีเพียงเท่านี้ ขอให้ป้อนและเก็บไว้ในไฟล์ WINDOW.ICL

โครงงานหน้าต่าง 1

สำหรับการทดลองใช้งานในช่วงแรกนี้จะเป็นแต่เพียงให้เปิดหน้าต่าง ปิดหน้าต่าง และให้ผู้ใช้กำหนดหน้าต่างตามต้องการ ดังโปรแกรมในรูป A,B,C และ M1

ขอให้ดูในรูป A ก่อนครับ เป็นการกำหนดค่าคงตัวต่าง ๆ ได้กำหนดหน้าต่างไว้เพียง 4 หน้าต่าง และได้กำหนดกรอบหน้าต่างไว้ 5 กรอบ คือ ที่กำหนดให้แก่ wndSize ค่าแรกเป็นของ wnd[0] คือจอภาพ จึงกำหนดขนาดของหน้าต่างไว้เต็มที่ ค่าที่สองเป็นของ wnd[1] ได้กำหนดไว้ในลักษณะที่เมื่อตีกรอบแล้วจะเหลือช่องหน้าต่างเพียงบรรทัดเดียว หน้าต่างสุดท้ายเป็นของ wnd[4] ได้กำหนดไว้เต็มที่เช่นกัน แต่จะให้ผู้ใช้กำหนดขนาดใหม่



รูปที่ 7 รหัสอักขระที่เป็นกรอบภาพ

A

```
PROGRAM  DemoPointer1(input,output);
USES     crt;

(*-----------------  CONSTANT  for WINDOW.ICL  -----------*)
CONST    MAXWNDW = 4;  MAXSTCK = 10;
TYPE     edgeT   = ARRAY[1..4] OF Byte;
         WindowSizeT = ARRAY[0..MAXWNDW] OF edgeT;
CONST    wndSize : WindowSizeT =
          ((1,80,1,25), (1,80,1,3), (1,80,10,24),
           (50,70,1,15), (1,80,1,25));
{$I WINDOW.ICL}
(*-----------------------------------------------------------*)
TYPE     promptT  = ARRAY[1..MAXWNDW] OF String[13];
CONST    promp    : promptT =
          ('FIRST WINDOW ','SECOND WINDOW','THIRD WINDOW','');
```

M1

```
BEGIN   { MAIN }
  SetUpWindow;

  WriteLn('MAIN SCREEN');
  repeat  readln(astr)  until  astr = '';

  OpenWnd1to(3);

  CreateUserWindow;

  CloseWndNto1(4);

END.
```

B

```
PROCEDURE   OpenWnd1to(n : Byte);
VAR   I : Integer;
BEGIN
  FOR  I := 1 TO n  DO
    BEGIN
    OpenWindow(wnd[I], I);
    Write(promp[I]);  if i > 1 then writeln;
    repeat  readln(astr) until astr = '' ;
    END;
END;

PROCEDURE  CloseWndNto1(n : Byte);
BEGIN
  FOR  n := n DOWNTO 1  DO
    BEGIN  CloseWindow;  ReadLn(astr)  END
END;
```

C

```
PROCEDURE  CreateUserWindow;
VAR  x1,x2,y1,y2 : Byte;
BEGIN
  WriteLn('Enter x1 x2 y1 y2  of your window');
  ReadLn(x1,x2,y1,y2);
  WITH  wnd[4]^  DO
    BEGIN  edge[1] := x1;  edge[2] := x2;
           edge[3] := y1;  edge[4] := y2   END;
  DrawFrame(wnd[4],wnd[4]);
  OpenWindow(wnd[4], 4);
  writeln('THIS IS YOUR WINDOW');
  repeat  readln(astr)  until  astr = '';
END;
```

ค่าคงตัวส่วนที่กล่าวมานี้จะใช้ในไฟล์ WINDOW.ICL ด้วย จึงตีกรอบไว้เป็นที่สังเกต และได้ Include ไฟล์นี้ตรงส่วนล่างของกรอบนี้ ส่วนค่าคงตัวที่เหลือเป็นชื่อหน้าต่างที่จะใช้ในแต่ละโปรแกรม ซึ่งอาจต้องการเปลี่ยน

โปรแกรมหลัก

โปรแกรมหลักอยู่ในรูป M1 ดำเนินการดังกล่าวมา คือเตรียมหน้าต่างด้วย SetUpWindow เสร็จแล้วให้พิมพ์คำว่า MAIN SCREEN แล้วให้ผู้ใช้ป้อนข้อมูลใดก็ได้เล่น ๆ คือ ไม่ได้นำไปดำเนินการใดจนกว่าจะกดคีย์ ENTER ที่ต้นบรรทัด

ต่อไปให้เปิดหน้าต่าง 3 หน้าต่างด้วย Openwnd1to ซึ่งจะให้ผู้ใช้ป้อนข้อมูลเล่น ๆ ทีละหน้าต่างอีกเช่นกัน แล้วให้ผู้ใช้เปิดหน้าต่างตามขนาดที่ต้องการด้วย CreateUserWindow จบด้วยการปิดหน้าต่างให้เห็นทีละหน้าต่างด้วย Close WndNtol ซึ่งมีรายละเอียดของแต่ละโปรแกรมดังจะกล่าวต่อไปคือ

OpenWnd1to

โปรแกรมนี้อยู่ในรูป B เพียงแต่เรียกใช้ OpenWindow เปิดหน้าต่างทีละหน้าต่าง แสดงชื่อหน้าต่าง แล้วให้ผู้ใช้ป้อนข้อมูลเล่น ๆ

CreateUserWindow

เปิดหน้าต่างตามขนาดที่ผู้ใช้ต้องการ อยู่ในรูป C เริ่มต้น โดยให้ผู้ใช้ป้อนค่าโคออร์ดิเนต นำมาให้เป็นค่ากรอบของ wnd[4] เสร็จแล้วให้สร้างกรอบต่อด้วยการเปิดหน้าต่างตามที่ได้เคยกล่าวมา

CloseWndNto1

ปิดหน้าต่างทีละหน้าต่าง โดยให้กดคีย์ ENTER ผ่านไป เพื่อให้เห็นภาพว่าปิดได้อย่างไร เพราะเป็นเพียงการทดลอง

สำหรับในฉบับนี้ผมขอจบเพียงเท่านี้ก่อน โครงงานในช่วงที่สองและที่สามขอนำไปกล่าวในฉบับหน้าครับ